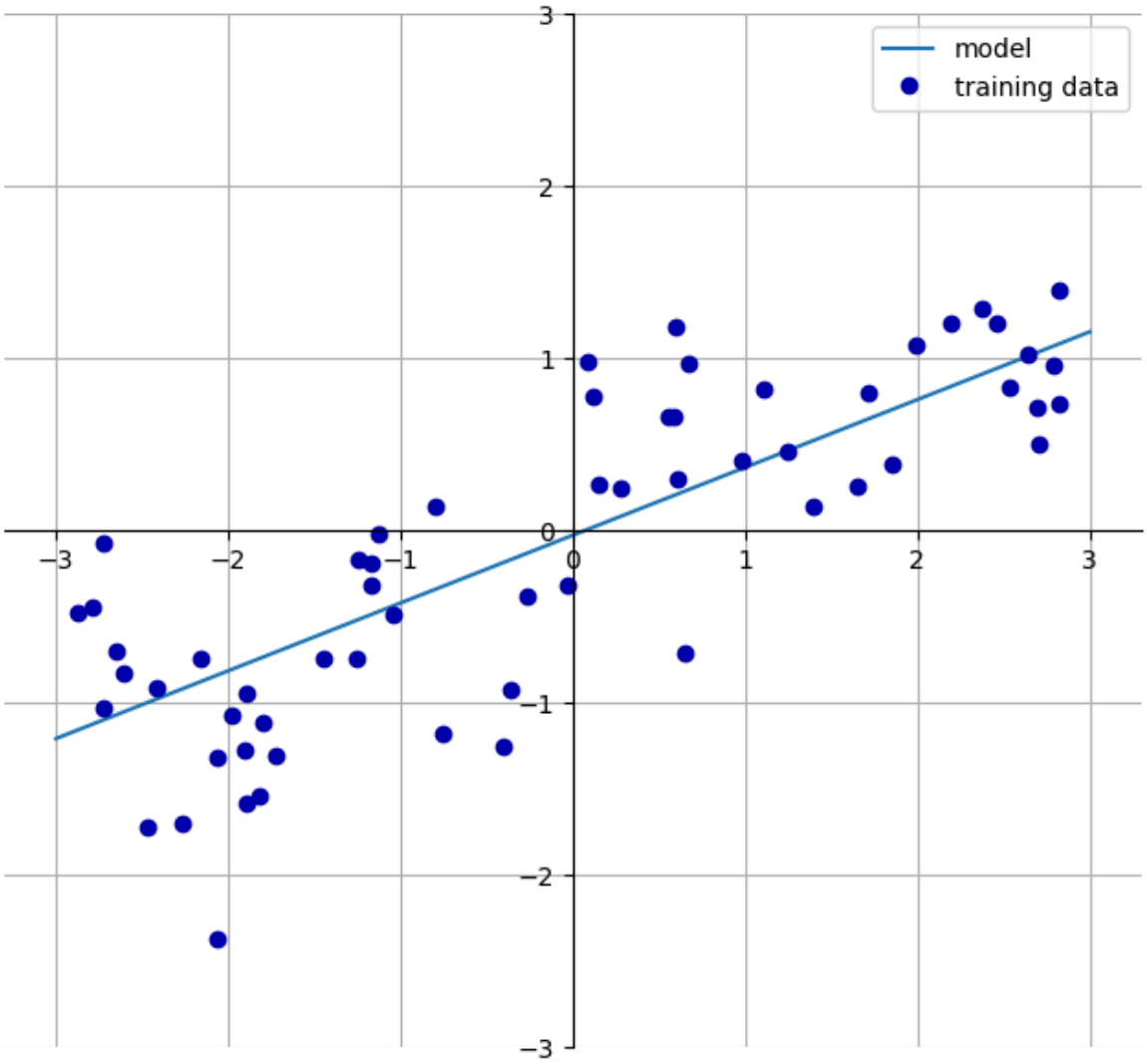
In [3]:
```
# LinearModels For Regression
# 线性模型预测公式： ŷ = w[0] * x[0] + w[1] * x[1] + ... + w[p] * x[p] + b
# x[0]到x[p]表示单个数据点的特征
# w和b是模型学习到的参数，ŷ是模型做出的预测
# 对于具有单个特征的数据集，这可以表示为： ŷ = w[0] * x[0] + b
import mglearn
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
mglearn.plots.plot_linear_regression_wave()
# 线性回归模型的特点
# 对于单个特征，预测为一条直线
# 对于两个特征，预测为一个平面
# 在使用更多特征时，预测为一个超平面

from sklearn.linear_model import LinearRegression
X, y = mglearn.datasets.make_wave(n_samples=60)
X_train, X_test, y_train, y_test = train_test_split(X, y ,random_state=42)
lr = LinearRegression().fit(X_train, y_train)
# 斜率参数（w），也称为权重或系数，存储在coef_中
# 偏移量或截距（b）存储在intercept_中
print("lr.coef_: {}".format(lr.coef_))
print("lr.intercept_: {}".format(lr.intercept_))
# 训练集精度
print("Training set score: {:.2f}".format(lr.score(X_train, y_train)))
# 测试集精度
print("Test set score: {:.2f}".format(lr.score(X_test, y_test)))
# 0.66的R²效果不佳，训练集和测试集的得分非常接近，这意味着可能存在欠拟合
# 对于这个一维数据集，过拟合的风险很小，因为模型非常简单（或受限）。
# 对于高维数据集，线性模型性能更好，过拟合的可能性也更高。

# 波士顿房价回归模型
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y ,random_state=0)
lr = LinearRegression().fit(X_train, y_train)
print("Training set score: {:.2f}".format(lr.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lr.score(X_test, y_test)))
# 训练集和测试集性能之间的差异是过拟合的明显迹象
```

```
w[0]: 0.393906  b: -0.031804
lr.coef_: [0.39390555]
lr.intercept_: -0.031804343026759746
Training set score: 0.67
Test set score: 0.66
Training set score: 0.95
Test set score: 0.61
```



In [4]:
```
# 岭回归也是一种线性回归模型
# 在岭回归中，我们希望系数的大小尽可能小，w的所有条目应接近于零
# 直观上，这意味着每个特征对结果的影响应尽量小同时
# 这个约束是所谓的正则化的一个例子
# 正则化意味着明确限制模型，以避免过拟合
# 岭回归使用的特定类型被称为L2正则化
from sklearn.linear_model import Ridge

ridge = Ridge().fit(X_train, y_train)
print("Training set score: {:.2f}".format(ridge.score(X_train, y_train)))
print("Test set score: {:.2f}".format(ridge.score(X_test, y_test)))
# 岭回归的训练集得分低于线性回归的得分，而测试集得分则更高
# 模型复杂度降低意味着训练集上的性能变差，但泛化能力更强
# 可以通过alpha参数指定模型对简单性与训练集表现的重视程度
# 在之前的示例中，我们使用了默认参数alpha=1.0
# alpha的最佳设置依赖于我们使用的特定数据集
# 增加alpha会使系数更趋向于零
# 这会降低训练集的性能，但可能有助于提高泛化能力
```

```
Training set score: 0.89
Test set score: 0.75
```

In [5]:
```
ridge10 = Ridge(alpha=10).fit(X_train, y_train)
print("Training set score: {:.2f}".format(ridge10.score(X_train, y_train)))
print("Test set score: {:.2f}".format(ridge10.score(X_test, y_test)))
```

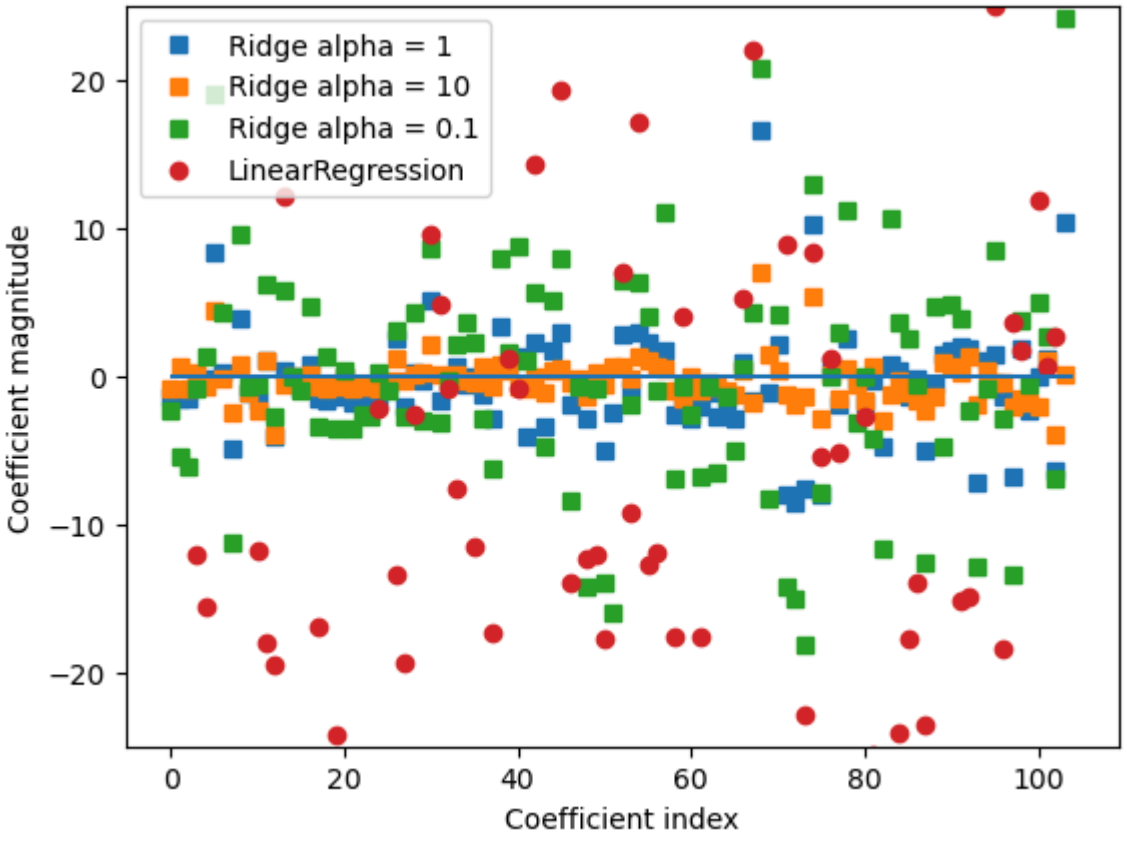
```
Training set score: 0.79
Test set score: 0.64
```

In [9]:
```
ridge01 = Ridge(alpha=0.1).fit(X_train, y_train)
print("Training set score: {:.2f}".format(ridge01.score(X_train, y_train)))
print("Test set score: {:.2f}".format(ridge01.score(X_test, y_test)))
# alpha=0.1的效果很好，可以尝试进一步减小alpha以改善泛化能力
# 通过检查不同alpha值模型的coef_属性，可以定性了解alpha如何改变模型
# 更高的alpha意味着模型更受限
# 在高alpha值时，coef_的条目的绝对值会比低alpha值时更小
# 以下图形证实了这一点
```

```
Training set score: 0.93
Test set score: 0.77
```

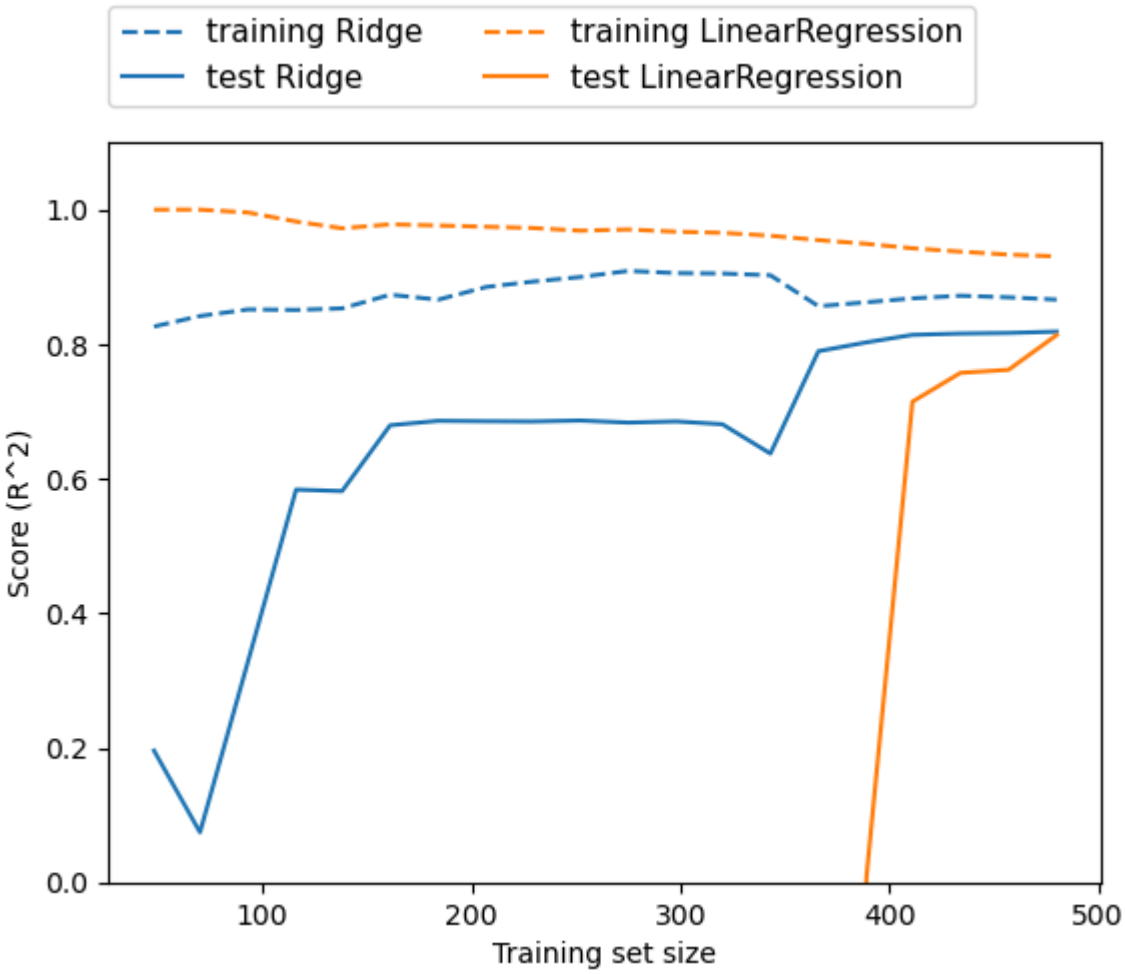
In [11]:
```
plt.plot(ridge.coef_, 's', label = "Ridge alpha = 1")
plt.plot(ridge10.coef_, 's', label = "Ridge alpha = 10")
plt.plot(ridge01.coef_, 's', label = "Ridge alpha = 0.1")
plt.plot(lr.coef_, 'o', label = "LinearRegression")
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")
plt.hlines(0, 0, len(lr.coef_))
plt.ylim(-25, 25)
plt.legend()
```

Out[11]: <matplotlib.legend.Legend at 0x1b3b37d1970>



In [73]:
```
# 理解正则化影响的另一种方法是固定alpha的值，改变可用的训练数据量
# 对波士顿住房数据集进行子采样
# 并在不断增大的子集上评估了线性回归和岭回归（alpha=1）的性能
# 显示模型性能与数据集大小的关系的图称为学习曲线
mglearn.plots.plot_ridge_n_samples()
# 对于所有数据集大小，岭回归和线性回归的训练得分都高于测试得分
# 由于岭回归是正则化的，其训练得分始终低于线性回归的训练得分
# 然而，岭回归的测试得分更好，特别是在数据的较小子集上
```

```
# 当数据点少于400时，线性回归无法学到任何东西
# 随着可用于模型的数据量不断增加，两种模型的性能都有所提高
# 最终线性回归与岭回归的表现持平
# 在足够的训练数据下，正则化变得不那么重要
# 并且在数据足够的情况下，岭回归和线性回归的性能将相同
# （在使用完整数据集时发生这种情况只是巧合）
```



In [13]:
```
# Lasso回归
# 岭回归的另一种正则化线性回归的替代方法是Lasso
# Lasso也将系数限制在接近于零的范围内，称为L1正则化
# L1正则化的结果：一些系数正好为零这意味着模型，完全忽略某些特征
# 部分系数为零通常使模型更易于解释，并且揭露模型中最重要的特征
from sklearn.linear_model import Lasso
import numpy as np
lasso = Lasso().fit(X_train, y_train)
print("Training set score:{:.2f}".format(lasso.score(X_train, y_train)))
print("Test set score:{:.2f}".format(lasso.score(X_test, y_test)))
print("Number of features used:{}".format(np.sum(lasso.coef_ != 0)))
```

```
Training set score:0.29
Test set score:0.21
Number of features used:4
```

In [15]:
```
# Lasso在训练集和测试集上的表现都相当差，这表明处于欠拟合状态
# 并且它仅使用了105个特征中的4个
# 与岭回归类似，Lasso也有一个正则化参数alpha
# 在前面的示例中，默认使用了alpha=1.0
# 为了减少欠拟合，可以尝试减小alpha
# 同时，还需要增加max_iter（最大迭代次数）的默认设置
lasso001 = Lasso(alpha=0.01,max_iter=100000).fit(X_train, y_train)
```

```
print("Training set score:{:.2f}".format(lasso001.score(X_train, y_train)))
print("Test set score:{:.2f}".format(lasso001.score(X_test, y_test)))
print("Number of features used:{}".format(np.sum(lasso001.coef_ != 0)))
```

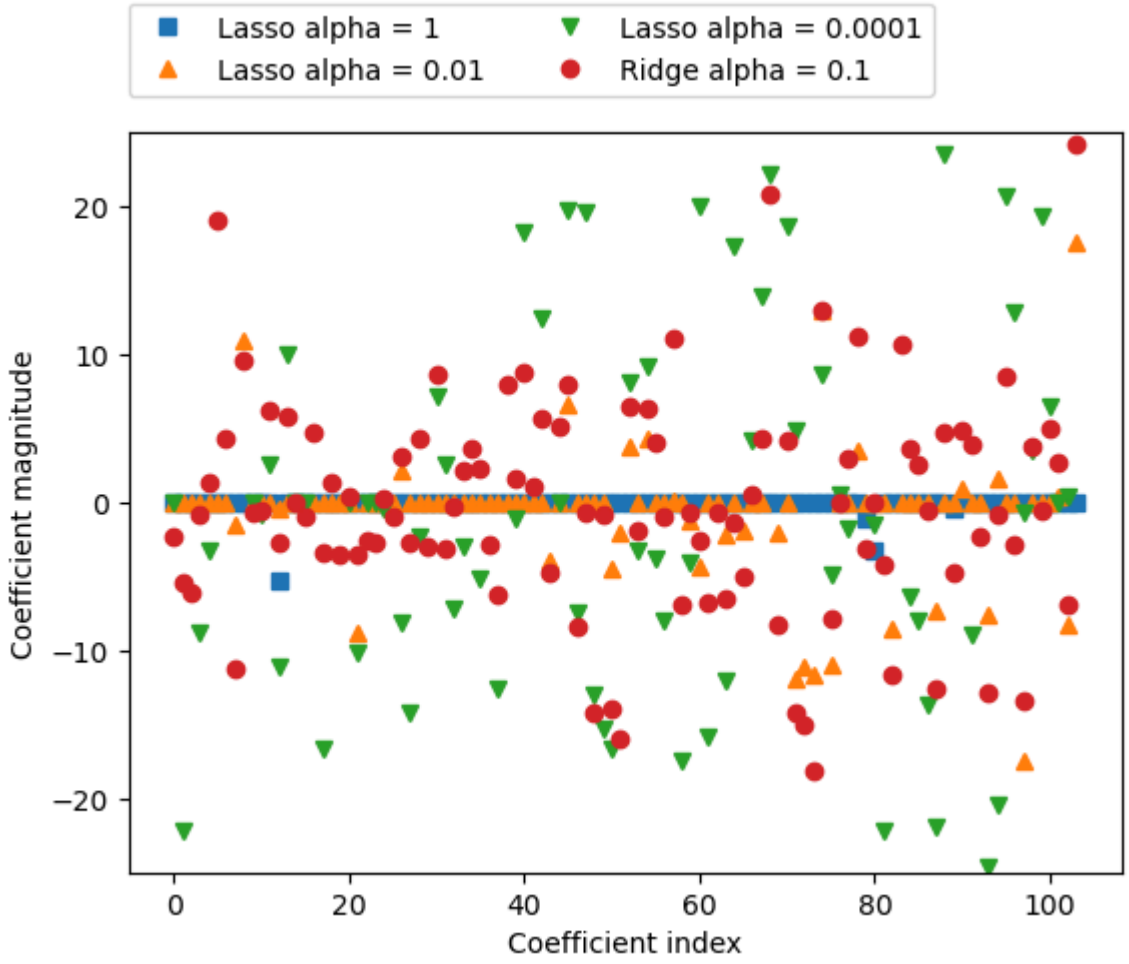
```
Training set score:0.90
Test set score:0.77
Number of features used:33
```

In [19]:
```
# 较低的 alpha 值能够拟合更复杂的模型，在训练集和测试集上的表现更好
# 其性能略优于使用 Ridge，并且仅使用了 105 个特征中的 33 个
# 如果将 alpha 设置得过低，就会失去正则化的效果，最终导致过拟合
# 结果类似于 LinearRegression
lasso00001 = Lasso(alpha=0.0001,max_iter=100000).fit(X_train, y_train)
print("Training set score:{:.2f}".format(lasso00001.score(X_train, y_train)))
print("Test set score:{:.2f}".format(lasso00001.score(X_test, y_test)))
print("Number of features used:{}".format(np.sum(lasso00001.coef_ != 0)))
```

```
Training set score:0.95
Test set score:0.64
Number of features used:96
```

In [23]:
```
# 绘制不同模型系数图
plt.plot(lasso.coef_, 's', label = "Lasso alpha = 1")
plt.plot(lasso001.coef_, '^', label = "Lasso alpha = 0.01")
plt.plot(lasso00001.coef_, 'v', label = "Lasso alpha = 0.0001")
plt.plot(ridge01.coef_, 'o', label = "Ridge alpha = 0.1")
plt.legend(ncol=2, loc=(0,1.05))
plt.ylim(-25, 25)
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")
# 当alpha=1时，大多数系数为零，而且剩下的系数也较小
# 当alpha降低到0.01时，大部分特征的系数为零
# alpha降为0.0001时，模型几乎没有正则化，系数大多为非零且数值较大
# 作为对比，最佳的Ridge解决方案alpha=0.1，所有系数均非零
# 与Lasso模型alpha=0.01有类似的预测性能
# 在实际操作中，通常在这两种模型之间优先选择Ridge回归
# 如果特征数量庞大且只期望少数特征是重要的，Lasso可能是更好的选择
# 因为Lasso仅会选择部分输入特征，因此会提供一个更易理解的模型
```

Out[23]:
```
Text(0, 0.5, 'Coefficient magnitude')
```


Lasso alpha = 1
Lasso alpha = 0.01
Lasso alpha = 0.0001
Ridge alpha = 0.1
Coefficient magnitude
20
10
0
−10
−20
0
20
40
60
80
100
Coefficient index